NASA-TM-111496

7N-64-TM
47993

A Model of Sequence Extrapolation

PHILIP LAIRD
ARTIFICIAL INTELLIGENCE RESEARCH BRANCH

RON SAUL
RECOM TECHNOLOGIES, INC.

PETER DUNNING
AMPEX CORPORATION

NASA Ames Research Center
Artificial Intelligence Research Branch
M/S 269-2
Moffett Field, CA 94035-1000
Technical Report FIA-93-12

May, 1993

A Model of Sequence Extrapolation

Philip Laird[*] A.I. Research Branch
Ronald Saul[†] Recom Technologies, Inc.
NASA Ames Research Center
Moffett Field, CA 94035-1000 (U.S.A.)

Peter Dunning
Ampex Corporation
Redwood City, CA. 94061

Abstract

We study sequence extrapolation as an abstract learning problem. The task is to learn a stream—a semi-infinite sequence $\langle s_1, s_2, \ldots, s_n, \ldots \rangle$ of values all of the same data type—from a finite initial segment $\langle s_1, s_2, \ldots, s_m \rangle$. We assume that all elements of the stream are of the same type (e.g., integers, strings, etc.). In order to represent the hypotheses, we define a language for streams called *elementary stream descriptions* and present an algorithm that learns in the limit elementary streams over an extensive family of data types. The complexity of the algorithm depends on the type and on a stream property known as the *delay.* In general the complexity is exponential or worse, but for streams with bounded delay over freely generated types the algorithm runs in time polynomial in the size of the examples. Sample size analysis is difficult, but for streams of delay 2 over applicative pairs, we calculate exactly the sample size required in the worst case to identify the stream uniquely. This bound helps explain why sequence extrapolation requires so few examples compared to statistical learning.

Introduction

Learning from experience to predict sequences of discrete symbols is a fundamental class of learning problems. Characteristic of such problems is that the order of the examples is an integral part of the learning problem; hence concept-learning methods based on independent random sampling do not apply. In this paper we consider the problem of *sequence extrapolation*, where the task is to form a rule for computing the n'th value of the sequence, given a finite number of its predecessors. As a motivating example, try to state a simple rule that accounts for the following sequence of strings and use it to predict the next string: `*?`, `!**??`, `!!*!**???`, `!!!*!!*!**?????`, ...

Extrapolation problems apply to many types of data: strings, integers, trees, alphabetic letter sequences, and so on. One finds that many of the same techniques for finding patterns apply regardless of the data type. Another feature of extrapolation problems is how few examples seem to be required in order to determine the rule. For example, sequences governed by polynomial functions $f(n) = \sum_{i=0}^{k} c_i n^i$ are learnable after only $k+1$ values. This stands in contrast to the very large sample sizes required for statistical learning problems.

The main result of this paper is an algorithm that extrapolates many sequences (including the example above) over a large class of data types. Easily understood and implemented, the algorithm improves the assortment of *ad hoc* techniques used heretofore. We have proved the correctness of the algorithm and studied its complexity. Our analysis also explains in general terms what properties determine the minimum number of examples necessary to identify the rule.

While not currently an active research area, sequence extrapolation has in the past been the subject of research for psychologists and computer scientists. Psychologists have examined in some detail the way humans infer rules for *Thurstone sequences*—sequences of alphabetic letters (e.g., ``A C B D C E...'') in which the alphabetic ordering provides the fundamental relationship from which the rule is formed. Programs that implement models of human performance on this task have also been used to test cognitive theories (e.g., [6]).

Over the years a number of programmers have tried to write powerful sequence extrapolation programs. For example, Pivar and Finkelstein [10] described a Lisp program that extrapolates integer sequences based on the so-called *method of differences.* Marcel Feenstra (cited in [2]) later extended the method with some clever heuristics and was able to achieve an "IQ" of about 160 on a published test. Evidently the method of differences is promising for extrapolating integer sequences but hard to extend to data types other than integers.

Noteworthy work in artificial intelligence on sequence extrapolation includes the little known but interesting thesis by Persson [9] and the paper by Hedrick [5]. Persson's approach was to seek a method for automatically constructing a new representation in which the problem reduces to a known or easy case. Hedrick used semantic nets to chart relationships among objects in an effort to find a method applicable to sequences of objects over

[*]Email: `laird@ptolemy.arc.nasa.gov`
[†]Email: `saul@ptolemy.arc.nasa.gov`

To appear in COLT '93 Proceedings.

several types. Dietterich and Michalski [3] described a program called SPARC/E to play the game Eleusis, in which the next element in a sequence is a set of object, not a unique object. Later this work was extended and generalized in [8].

A recurring theme in past research is that sequence-extrapolation algorithms should be type independent, with a type-dependent part reflecting differences among data structures. Our results are perhaps the first to capture this intuition formally: Using concepts from algebraic type theory, we have separated the type-independent and type-dependent aspects and constructed an algorithm in which the type is a parameter. Extrapolation is not equally easy for all types, however, and our analysis shows that a fundamental property of types—whether or not they are freely generated—makes a big difference in the complexity of the algorithm. Also significant, however, is a property of the stream itself: the number of previous elements required to predict the next one. We call this the *delay*, and show that the complexity may be exponential in the delay.

Types

We want our extrapolation algorithm to apply to many data types. The types we define are universal algebras with some additional restrictions. For simplicity we consider only types of a single sort, but extension to the many-sorted case is straightforward.

Let $\mathcal{F}$ be a finite set of symbols representing operations, where each symbol $f \in \mathcal{F}$ is assigned a nonnegative integer (its *arity*), and the subset $\mathcal{F}_0$ of *atoms* (symbols of arity zero) is nonempty. The *set $\mathcal{E}$ of expressions generated by $\mathcal{F}$* is the free algebra generated by $\mathcal{F}_0$ and closed under the formal operations $\mathcal{F}$.

The operations of arity zero are called *atoms* or *generators.* For example, if f is an operation of arity 2 and a and b are atoms, then a, b, and $f(f(a,a),b)$ are a few of the expressions in $\mathcal{E}$.

A *type D over $\mathcal{F}$* is a family of congruences, i.e., equivalence classes on $\mathcal{E}$ compatible with the operations $\mathcal{F}$. Thus if d_i and e_i are equivalent expressions $(1 \leq i \leq n)$ and f is an n-ary operation, then $f(d_1,\ldots,d_n)$ is equivalent to $f(e_1,\ldots,e_n)$.

Examples of familiar data types are:

- *Strings* over the alphabet $A = \{a,b\}$, where the only operator (apart from constants a and b) is the associative binary function $\odot$ concatenating two strings. The two expressions $(a \odot b) \odot c$ and $a \odot (b \odot c)$ are equivalent strings since the string operation is associative.
- *Dotted pairs* over the set $A = \{a,b\}$. The operator $(\cdot)$ makes a pair out of two expressions. Typical expressions include a, b, $(b \cdot b)$, $(a \cdot (b \cdot a))$, etc.
- *Non-negative integers* generated from $\{0,1\}$ by the binary operators $+$ (addition) and $*$ (multiplication). Both operators are associative and commutative.
- *Thurstone domain:* The letters A, B, ..., Z together with the constructors *successor* and *predecessor*, e.g., *successor*(A) = B and *predecessor*(B) = A.

Let $\succeq$ be the following partial ordering on the expressions $\mathcal{E}$: the minimal elements are the generators, and if $y = f(x_1,\ldots,x_n)$ for some operation $f \in \mathcal{F}$, then $y \succeq x_i$ for $i \leq n$. This ordering relation extends to the elements of the type D: if x and y are expressions and $[x]$ and $[y]$ denote their equivalence classes, then $[y] \succeq [x]$ whenever $y \succeq x$. On D, $\succeq$ need not be a partial ordering.

Definition 1 A type D is said to be *constructive* if $\succeq$ on $\mathcal{E}$ extends to a partial ordering on D.

Strings with concatenation generated by a finite alphabet, the non-negative integers with addition generated by 0 and 1, and the set of ordered trees (lists) generated by a finite set of atoms under the action of the operations **cons** and **append** are all examples of constructive types. The non-negative integers with addition and multiplication are not a constructive type: $1 = 0+1$ and $0 = 1 \times 0$, so that $1 \succeq 0 \succeq 1$. Neither is the Thurstone domain, since ordinarily we think of A as the successor of Z; but by pairing each letter with an index counting the number of "wraparounds"—e.g., *successor*($\mathtt{Z}_1$) = $\mathtt{A}_2$, we can handle Thurstone sequences with contructive types.

The domain of dotted pairs—probably the simplest non-trivial constructive type—plays a fundamental role in our model. Combinators and ordered binary trees are common applications of this data structure.

A constructive type is said to be *freely generated* if each value can be constructed from the generators by exactly one sequence of operations. Pairs, for example, are freely generated, but strings and naturals are not: the string abc can be constructed $(ab)c$ or $a(bc)$, and 3 can be constructed as $(1+1)+1$ or $1+(1+1)$.

Finally, we assume that types are represented so as to be effectively distinguishable. For example, there can be no confusion between the integer 123 and the string "123".

Streams

A *stream* is a semi-infinite sequence $\langle s_1, s_2, \ldots \rangle$ of values, all of the same type. The first element s_1 is called the *head*, and the stream $\langle s_2, s_3, \ldots \rangle$ is called the *tail.*

The operations for a type D extend naturally to streams of the same type—e.g., if $S = \langle s_1, \ldots \rangle$ and $T = \langle t_1, \ldots \rangle$ are streams and $f(,)$ is an operation, then $f(S,T) = \langle f(s_1,t_1),\ldots \rangle$ is a stream. The constant streams $\mathbf{a} = \langle a, a, \ldots \rangle$, where a is a generator, are stream operations of arity 0.

In order to formalize sequence extrapolation as learning a stream, we define a representation language for

$$\begin{array}{rcl} S & = & \langle a \mid T\rangle \\ T & = & [S', U] \\ S' & = & S \\ U & = & a \end{array} \qquad\qquad \begin{array}{rcl} F & = & \langle 1 \mid G\rangle \\ G & = & \langle 1 \mid H\rangle \\ H & = & F' + G' \\ F' & = & F \\ G' & = & G \end{array}$$

(a) (b)

Figure 1: Example elementary descriptions.

streams. In describing the language, we shall adopt the convention that uppercase letters (S, T, etc.) denote the names of streams, lowercase Roman letters without subscripts (a, b, etc.) denote atomic expressions, and Greek letters (α, β, etc.) denote arbitrary expressions. The i'th value of the stream A is denoted by a_i, and similarly for other stream symbols.

Definition 2 An *elementary description* (of type D) of a stream S is a block-structured list of one or more equations. The block has one of the following forms:

- (*initial-value form*) $S = \langle a \mid T\rangle$, where a is an atom of type D, and T is a stream name different from S. This equation is followed by an elementary description of T. (Intuitively the atom a represents the head, and T the tail stream.)
- (*functional form*) $S = f(T_1, \ldots, T_k)$, where f is an operation of arity $k \geq 0$, and the symbols T_i are distinct stream names different from S. This equation is followed by elementary descriptions of $T_1, \ldots, T_k$. (Intuitively this represents the stream S as a function f of the streams T_i.)
- (*equality form*) $S = U$, where U is the name of a stream in the parent hierarchy of S (see below). (This defines the stream S recursively in terms of another stream U.) ▮

In the initial-value form, S is said to be the *parent stream* of T; likewise in the function form S is the parent stream of each T_i ($1 \leq i \leq k$). The *parent hierarchy* of a description T (or T_i, etc.) consists of S, the parent of S, its parent, and so on. A stream U named in an equality definition of S must belong to the parent hierarchy of S.

As a simple example, the single equation $S = a$ is an elementary description. (Recall that atoms are functions of arity zero.) It denotes the constant stream $\langle a, a, a, \ldots\rangle$.

The description in Figure 1(a) represents the stream $S = \langle a, [a, a], [[a, a], a], \ldots\rangle$ of type *pair*. Over the naturals with addition, the description in Figure 1(b) describes the Fibonacci sequence, $F = \langle 1, 1, 2, 3, 5, \ldots\rangle$.

The language can be made more economical by allowing backward references to parent streams in the initial-value and functional forms. We can then eliminate one equation from the stream in Figure 1(a). writing $T = [S, U]$. We formally adopt the more restricted form in order to simplify the reasoning about the language. On occasion, where formality is less of an issue than clarity, we may use such "improper" backreferences.

Before we provide a semantics for elementary descriptions, we must place additional restrictions on the language. Otherwise we may suffer circular descriptions such as $S = f(T)$, $T = S$.

Definition 3 Let S be an elementary description for a stream, and let W be a stream name defined within the block S. The *relative delay* $\Delta(S, W)$ of W relative to S is defined as follows:

- $\Delta(S, S) = 1$;
- If $S = \langle a \mid T\rangle$, then $\Delta(S, W) = 1 + \Delta(T, W)$.
- If $S = f(T_1, \ldots, T_k)$ ($k > 0$) and W is defined within the block T_i, then $\Delta(S, W) = \Delta(T_i, W)$. ▮

Definition 4 Let S be an elementary description for a stream. The *delay* in the description S is $\max\{\Delta(S, W) \mid W \text{is defined within } S\}$.

Definition 5 An elementary description for a stream S is called *proper* if for every equality form $S_i = U$ in the block, $\Delta(U, S_i) > 1$.

Intuitively the delay is the number of previous values upon which each successive value depends, plus one. (The "plus one" is for technical reasons.) In the example in Figure 1(a) above, the delay in S is 2. For the family of Fibonacci numbers defined in Figure 1(b), the delay of F is 3 since $\Delta(F, H) = 3$ is maximum over all the symbols defined in the block F. Both are proper descriptions. The description $S = f(T)$, $T = S$ is not proper since $\Delta(S, T) = 1$.

Proper elementary descriptions have a well-defined semantics that assigns as a model a stream $[\![S]\!]$ to every stream symbol S:

- If $S = \langle a \mid T\rangle$, then $[\![S]\!]$ is the stream whose head is a and whose tailstream is $[\![T]\!]$.
- If $S = f(T_1, \ldots, T_k)$, then $[\![S]\!]$ is the stream $f([\![T_1]\!], \ldots, [\![T_k]\!])$.
- If $S = U$ (an equality form), then $[\![S]\!]$ is $[\![U]\!]$.

Theorem 6 *A proper elementary description S has a unique model stream. If the description S is not proper, it may have one model, no models, or many models.*

(The proof can be made by induction or with a fixpoint argument.)

For example, in Figure 1(a),

$$\begin{aligned}
[\![S]\!] &= \langle a, t_1, \ldots, t_n, \ldots\rangle \\
[\![T]\!] &= \langle t_1, \ldots, t_n, \ldots\rangle \\
&= \langle [s'_1, u_1], \ldots, [s'_n, u_n], \ldots\rangle \\
[\![S']\!] &= \langle s'_1, \ldots, s'_n, \ldots\rangle \\
&= [\![S]\!] \\
[\![U]\!] &= \langle u_1, \ldots, u_n, \ldots\rangle \\
&= \langle a, a, \ldots\rangle
\end{aligned}$$

Over the non-negative integers the description $X = Y + Z, Y = X, Z = 0$ can be modeled by any stream X over the type. But if we change the description of Z to $Z = 1$, then X has no model over the type.

Finally we call a stream S of elements of type D *an elementary stream* of type D if there is a proper elementary description whose model is S. ES denotes the class of all elementary streams. The delay of an elementary stream S is the least k such that S is representable by an elementary description with delay k. $ES(k)$ denotes the family of elementary streams with delay k.

Henceforth all streams under discussion will be presumed to be elementary and all descriptions, proper.

Summarizing, we have defined a limited family of data types, a language for describing streams over those types, and a semantics for the language. We shall model sequence extrapolation as the problem of learning a stream description, given an initial segment of the stream. The length of the initial segment is the samples size.

How Many Examples Are Needed?

Recall the stream $S = \langle a, [a,a], [[a,a],a], \ldots\rangle$ of type *pair* used in example above. Suppose we are told that this stream is in $ES(2)$. How many descriptions of delay two or less are consistent with these first three values[1]? With a bit of reflection we can convince ourselves that the only consistent description is that given in Figure 1(a) or one that is semantically equivalent. There is no stream S' of delay two beginning with these three values that later differs from S on some other value. If, however, we remove the restriction that the delay is two, then many different descriptions beginning with these same three values can be given.

Just as for statistical concept learning, the complexity of the learning problem for streams depends on both the minimum sample size and the search for a consistent description. Unlike statistical concept learning, however, the minimum sample sizes seem to be *much* smaller. While working with sequence extrapolation, the authors have often been surprised by how few examples are required to determine a stream description of smallest delay, regardless of type. Evidently the sequential ordering of the examples severely constrains the number of candidate descriptions. For statistical concept learning, the combinatorial dimension [1, 4] of the hypothesis family characterizes quite well the way examples constrain the number of consistent concept descriptions. Finding an equally satisfying way to characterize these constraints for sequence learning presents a challenge for computational learning theory.

[1]We shall use the terms *values* and *examples* interchangeably.

Definition 7 Let $\mathcal{S}$ be a class of elementary stream descriptions. We say that $\mathcal{S}$ is *m–distinguishable* if, for any two inequivalent stream descriptions A and B in $\mathcal{S}$, there exists an i, $1 \leq i \leq m$ such that $a_i \neq b_i$. $\mathcal{S}$ is said to be *exactly m–distinguishable* if it is m–distinguishable but not $(m-1)$–distinguishable.

If the class $\mathcal{S}$ is m–distinguishable, then any algorithm A that finds an elementary description consistent with the first m values of a stream can be turned into a learning algorithm for the class. The algorithm first obtains m values and then calls A.

We have found the task of calculating the minimum sample size to be difficult. For the special case of *pair* types, we have the following modest but interesting result:

Theorem 8 *Over pairs let $ES(2;h)$ be the family of elementary streams of delay at most two for which the first element has height h or less.*[2] *The family $ES(2;h)$ is exactly $(h+3)$-distinguishable.*

A sketch of the proof is appended.

Note that this is an *exact bound* on the worst-case sample size. $h+3$ examples suffice for all delay-2 streams, and for every h one can exhibit a stream whose first element has height h and for which there are at least two descriptions consistent with the first $h+2$ values.

As a corollary, if one begins the extrapolation process with the n'th value of the stream instead of with the first, the required sample size is greater by a factor of only a polynomial in n. The reason is that, whereas the sizes of successive elements can grow exponentially, their heights increase only linearly; and the theorem shows that the height is what determines the sample size.

Based on our experiences, we conjecture that *over pairs, the family $ES(k;h)$ of elementary streams with delay at most k is $(1+k+h)$-distinguishable, where h is the height of s_{k-1}.*

The Extrapolation Algorithm

We present an algorithm that searches for elementary descriptions consistent with the examples seen so far.

[2]Viewing pairs as binary trees, we say that an atom has height zero, and the pair $[\alpha,\beta]$ has height $h([\alpha,\beta]) = 1 + \max(h(\alpha), h(\beta))$.

Our algorithm keeps many active hypotheses for the input stream and eliminates hypotheses as soon as they conflict with the input examples.

The key to the algorithm's efficiency is the following recursive representation for sets of hypotheses. Let S be an input stream we want to identify. A ***hypothesis*** for S has a type, a confirmed length (equal to the number of elements of S with which the hypothesis has agreed so far), and one of the following formats:

- ***initial-value*** format: the hypothesis has the form $\langle s_1 \mid \mathcal{H}\rangle$, where s_1 is equal to the atomic head of the stream and $\mathcal{H}$ is the name of an ***induction space*** (defined below) for the set of possible tail streams.
- ***functional*** format: the hypothesis has the form $f(\mathcal{H}_1, \ldots, \mathcal{H}_k)$, where f is an operation of arity $k \geq 0$ and the $\mathcal{H}_i$ are the names of induction spaces.
- ***equality*** format: the hypothesis is simply $\mathcal{U}$, the name of an induction space.
- ***unknown*** format: This is a stream variable whose confirmed length is always zero. It is a placeholder representing a stream whose name is assigned but whose first element has not yet been received. We denote this hypothesis by the symbol $\Box$.

One hypothesis has exactly one of these formats; but for a stream there may be several hypotheses with different formats. An ***induction space*** (or ***space***) is a collection of heteromorphic hypotheses for the same stream. An induction space shares the same name, type, and confirmed length with the stream it represents. In addition it is assigned a parent induction space.

The algorithm receives the sequence $\langle s_1, s_2, \ldots\rangle$ of values of a stream S and incrementally constructs an induction space of consistent descriptions of S. The algorithm also calls upon an external halting criterion that determines when to stop and declare success.

Hypotheses for S are developed in the induction space called *MAIN-SPACE*, initially consisting of only $\Box$. In order to ensure that only proper descriptions occur as hypotheses, the algorithm assigns a relative delay $\Delta(S, T)$ to each stream T named in the induction space S or in any of its subspaces, in accordance with Definition 3. Finally, this algorithm assumes that the elements in the input stream are presented without any errors.

The output is an induction space of hypotheses consistent with the input stream, from which elementary descriptions may be extracted.

The Extrapolation Algorithm:

1. Initialize: *MAIN-SPACE* $:= \{\Box\}$. The confirmed length is zero and the name of the space is that of the input stream (here: S). Its relative delay is 1.
2. For $i := 1, 2, 3 \ldots$, until the termination condition is true: Let s_i be the next input value. Set *MAIN-SPACE* := *EXTEND*(*MAIN-SPACE*, s_i, i) and increment the confirmed length of *MAIN-SPACE* by one.
3. Output *MAIN-SPACE*.

The *EXTEND* Routines. Let $\mathcal{H}$ be a space representing a stream, s an input value, and $i \geq 1$. The procedure *EXTEND*$(\mathcal{H}, s, i)$ is:

For each $H \in \mathcal{H}$, $\mathcal{H} := (\mathcal{H} - \{H\}) \bigcup$ *EXTEND-HYP*$(H, s, i, \mathcal{H})$.

EXTEND replaces each hypothesis H in the space $\mathcal{H}$ by the set (perhaps empty) of all hypotheses that extend H to be consistent with the new value s.

The routine *EXTEND-HYP*$(H, s, i, \mathcal{H})$ is defined by cases on the form of H. For each possible form of H, it either discards H (if it is inconsistent with the next value s) or replaces it by all its extensions, obtained by recursively calling *EXTEND* for each sub-hypothesis that is part of H. To simplify the presentation we assume that the type is freely generated; for general types the algorithm is somewhat more complicated. (See discussion below.)

- Case: H is $\Box$.
  Return the following set of hypotheses, each with a confirmed length of one:
  - The initial-value stream $\langle s \mid \mathcal{H}'\rangle$, if s is an atom. The tailstream, represented by the new induction space $\mathcal{H}'$, is assigned a fresh name, a confirmed length of zero, and a relative delay one greater than that of $\mathcal{H}$; it is initialized to contain only the hypothesis $\Box$.
  - The name of every space $\mathcal{U}$ such that: (1) $u_1 = s$, (2) the relative delay of $\mathcal{U}$ is less than that of $\mathcal{H}$, and (3) $\mathcal{U}$ is in the parent hierarchy of $\mathcal{H}$.
  - The functional hypothesis $f(\mathcal{H}_1, \ldots, \mathcal{H}_k)$, $k \geq 0$, provided that[3] $s = f(x_1, \ldots, x_k)$, $f \in \mathcal{F}$, $s \neq x_i$ for $1 \leq i \leq k$. Each $\mathcal{H}_j$ is a new space initialized to contain only $\Box$ and immediately extended by calling *EXTEND*$(\mathcal{H}_j, x_j, 1)$. $\mathcal{H}$ is made the parent space of each $\mathcal{H}_j$. The relative delay of each $\mathcal{H}_j$ is equal to that of $\mathcal{H}$.
- Case: H is an initial-value hypothesis.
  Let $H = \langle a \mid \mathcal{H}'\rangle$; return the space $\{\langle a \mid$ *EXTEND*$(\mathcal{H}', x, i-1)\rangle\}$, or the empty space if this call to *EXTEND* returns the empty space. Increase the confirmed length of H by one.
- (Case: $H = \mathcal{U}$, another induction space)
  If $x = u_i$, return $\{H\}$ with a confirmed length of i; otherwise, return the empty set.
- (Case: H is a functional hypothesis)
  Let $H = f(\mathcal{H}_1, \ldots, \mathcal{H}_k)$, $k \geq 0$. If there exist no values $x_1, \ldots x_k$ such that $s = f(x_1, \ldots, x_k)$ and

[3]If the type is not freely generated, there may be many such hypotheses.

$s \neq x_i$ (for any i), return the empty set. Else let $s = f(x_1, \ldots, x_k)$; return, with a confirmed length of i, the hypothesis $f(EXTEND(\mathcal{H}_1, x_1, i), \ldots, EXTEND(\mathcal{H}_k, x_k, i)$.

Analysis

We summarize some results of our analysis. For simplicity let us continue limiting the discussion to freely-generated types.

Unlike most algorithms in computational learning theory, the extrapolation algorithm is sufficiently complex that a formal correctness proof is useful. Correctness means that, after receiving n input values, the main space contains only hypotheses consistent with the examples (*soundness*), and that if H is a hypothesis consistent with the input examples, H or an equivalent is represented in the main space (*completeness*).

We adopt the notation $x[i..j]$ for the subsequence $x_i, \ldots, x_j$ of values from the stream X. If $j < i$, it denotes the empty sequence.

The following definition makes precise what we mean when we say that an elementary description is "in" or "part of" an induction space. This is needed because induction spaces are not merely sets of descriptions.

Definition 9 Let $\mathcal{S}$ be an induction space, and let D be a proper elementary definition of a stream S_1, containing definitions for the symbols $S_1, S_2, \ldots, S_n$. We say that D is *part of* $\mathcal{S}$ when there is an injection σ mapping each symbol S_i into the spaces named in $\mathcal{S}$ such that $\sigma(S_1) = \mathcal{S}$ and, for $1 \leq i \leq n$,

- If $S_i = \langle v \mid S_j \rangle$, the hypothesis $\langle v \mid \mathcal{T} \rangle$ is one of the hypotheses in the space $\sigma(S_i)$, and $\sigma(S_j) = \mathcal{T}$.
- If $S_i = f(\ldots, S_j, \ldots)$, the hypothesis $f(\ldots, \mathcal{T}_j, \ldots)$ is in $\sigma(S_i)$, and for each j, $\sigma(S_j) = \mathcal{T}_j$.
- If $S_i = a$ (a constant), then the constant hypothesis a is in the space $\sigma(S_i)$.
- If $S_i = S_j$ (an equality reference), the equality hypothesis $\sigma(S_j)$ is in the set $\sigma(S_i)$.

An important detail is that, if $f(\mathcal{H}_1, \mathcal{H}_2, \ldots)$ is one of the hypotheses in a space $\mathcal{S}$, and H_1 is part of the space $\mathcal{H}_1$, and H_2 is part of the space $\mathcal{H}_2$, etc., then the description $S = f(H_1, H_2 \ldots)$ (followed by descriptions of H_1, H_2, and any other substreams) is a valid hypothesis. In other words, we may mix and match hypotheses freely from the spaces $\mathcal{H}_i$ without having to worry whether the combination is a valid elementary description. This is a result of the context-free property of elementary definitions: nothing in the block defining H_1 depends on, or refers to, any symbol in the block defining H_2 or other substream named in the functional hypothesis. We can build this observation into

Lemma 10 *If a description D is part of the induction space $\mathcal{S}$, then D is a proper elementary description.*

Definition 11 Let $\mathcal{I}$ be an induction space, and let $x[1..N]$ be the first N values of a stream X (for $N \geq 0$). We shall call $\mathcal{I}$ *consistent with* $x[1..N]$ if $N = 0$ and $\mathcal{I} = \{\Box\}$, or if $N > 0$ and every hypothesis in $\mathcal{I}$ is consistent with $x[1..N]$. A hypothesis H in $\mathcal{I}$ is *consistent with* $x[1..N]$ if:

- $H = \langle x_1 \mid \mathcal{H} \rangle$ and $\mathcal{H}$ is consistent with $x[2..N]$;
- $H = U$ (an equality reference), and for all $1 \leq i \leq N$, $x_i = u_i$;
- $H = a$ (a constant stream), and for all $1 \leq i \leq N$, $x_i = a$;
- $H = f(\ldots, \mathcal{H}_j, \ldots)$, $x[1..N] = f(\ldots, x_j[1..N], \ldots)$, and for all j, $\mathcal{H}_j$ is consistent with $x_j[1..N]$.

Lemma 12 (SOUNDNESS) *Let S be a stream whose elements are presented in sequence to the extrapolation algorithm. For all $N \geq 0$, after processing the N'th element,* MAIN-SPACE *is consistent with $s[1..N]$.*

The proof is essentially an induction on the inductive hypothesis that, if every H in a space $\mathcal{H}$ is consistent with the stream values $x[i..N-1]$, then every H' in EXTEND$(\mathcal{H}, x_N, N-i+1)$ is consistent with $x[i..N]$.∎

The idea behind the completeness argument is that, after the first k values of an input stream S have been presented, every consistent hypothesis with delay k or less, or some equivalent, is in the induction space for S.

Definition 13 Let S be an elementary stream description with delay D. The *r-approximation* $S(r)$ is defined for $r \geq 0$ as follows:

1. If $r = 0$, $S(r) = \Box$ (the unknown hypothesis);
2. If $r \geq D$, $S(r) = S$;
3. If $0 < r < D$ and
   - 3.1 $S = \langle v \mid T \rangle$, then $S(r) = \langle v \mid T(r-1) \rangle$.
   - 3.2 $S = f(T_1, \ldots, T_k)$, then $S(r) = f(T_1(r), \ldots, T_k(r))$. (Note: if $k = 0$, then $D = 1$, so preceding cases apply.)
   - 3.3 $S = U$, then $S(r) = U(r + \Delta(U, S))$.

Example: The 2-approximation to the description

$$\begin{aligned} S_1 &= \langle 1 \mid S_2 \rangle \\ S_2 &= S_1' + S_3 \\ S_1' &= S_1 \\ S_3 &= \langle 1 \mid S_2' \rangle \\ S_2' &= S_2 \end{aligned}$$

is $S_1(2) = \langle 1 \mid S_2(1) \rangle$, where $S_2(1) = S_1'(1) + S_3(1)$. $S_1'(1) = S_1(2)$ since $\Delta(S_1, S_1') = 1$. $S_3(1) = \langle 1 \mid S_2'(0) \rangle$, and $S_2'(0) = \Box$. These are the hypotheses corresponding to the definition of S_1 as they exist in the induction space after the algorithm has processed the first two values of S_1. After three values, $S_1(3) = S_1$ since the delay of S_1 is 3.

Lemma 14 (COMPLETENESS) *Let S be a stream in ES. For all $N \geq 0$, after the extrapolation algorithm has obtained and processed inputs $s[1..N]$, the N-approximation to every description of S is part of the* MAIN-SPACE.

The proof is an induction on the confirmed length of the induction space, based on the following inductive hypothesis: If $\mathcal{I}$ is a space for a stream X with confirmed length n and $H(n)$ is part of $\mathcal{I}$, then $H(n+1)$ is part of EXTEND$(\mathcal{I}, x_{n+1}, n+1)$. Since $\Box$ is the 0-approximation to every description of S, and for each input value the algorithm extends each approximation by one, the result follows. ▮

Theorem 15 (CORRECTNESS) *Let $S \in ES(k)$ be presented to the sequence extrapolation algorithm. There exists an integer $m \geq k$ such that after the first m values of S have been obtained and processed by the algorithm, the set of hypotheses with delay at most k that are part of* MAIN-SPACE *is semantically equivalent to the set of elementary descriptions of S.*

PROOF: We may verify that only proper hypotheses are introduced into the *MAIN-SPACE* by the algorithm. By Lemma 10 any hypothesis may be selected from any induction space to yield a syntactically valid stream description. Hence every hypothesis that is part of the *MAIN-SPACE* can be turned into an elementary description. The Soundness Lemma ensures that every description with delay $< k$ will eventually be eliminated, since such descriptions are necessarily inconsistent with S. For the same reason every hypothesis with delay k that does not describe S will eventually be eliminated. The Completeness Lemma says that every k-approximation to a description of S is part of the space. Among these are all descriptions with with delay k, since such descriptions are their own k-approximations. Since there are only finitely many hypotheses with delay k or less, the theorem follows. ▮

For a freely-generated type, extracting from an element $x = f(x_1, \ldots, x_i)$ its unique functional components x_i is a polynomial-time operation for most types. When it is, we can show:

Theorem 16 *For the family $ES(k)$ over a freely generated type, the algorithm processes input examples in time bounded by a polynomial in the total size of the examples and $\mathcal{O}(2^k)$.*

And in view of the sample-size result, we have:

Corollary 17 *The family $ES(2)$ over pairs is learnable in time polynomial in the size of the first $m = h(s_1) + 3$ examples.*

Non-freely Generated Types.

The extrapolation algorithm breaks input values s_i into functional components $s_i = f(s_{i_1}, \ldots, s_{i_k})$ in order to find functional hypotheses. When data types are not freely generated, there may be more than one way to do so—e.g., the integer 4 can be written $1+3$ or $2+2$ or $3+1$. In such cases the algorithm can be viewed as non-deterministic: if an external oracle were available to tell it which functional form $A+B$ was the "right" one, the algorithm would require no modification. Lacking such an oracle, however, the algorithm must consider all possible constructions.

Our approach is retain the single functional hypothesis $f(\ldots, H_i, \ldots)$ but replace each of the spaces H_i by a *tree of spaces* representing the ways of constructing the example. Consider the integer stream $\langle 3, 4, \ldots \rangle$, for example. These first two values actually represent *six possible pairs of examples* to a functional form like $A+B$: For the left halfstream $A = \langle a_1, a_2, \ldots \rangle$, a_1 could be 1 or 2 and a_2 could be 1, 2, or 3; and for $B = \langle b_1, b_2, \ldots \rangle$, the corresponding values of b_1 would be 2 or 1 and, for b_2, 3, 2, or 1. Thus instead of the single hypothesis of the form $A+B$, we must maintain six. Not all of these hypotheses will remain consistent with the subsequent examples.

We cannot give the full algorithm here, but the main point is that when the data type is not freely generated, the same algorithm applies with some additional bookkeeping. Naturally, however, the number of hypotheses grows much more rapidly than it does for freely generated types. Unless the type has *bounded multiplicity*—i.e., for every element there is a bound on the number of ways it can be factored into components by the basic operations—the extrapolation algorithm requires space superpolynomial in the size of the input values, in the worst case.

Naturally, this does *not* mean that extrapolating elementary streams of naturals, strings, and other non-freely generated types is an inherently hard problem: some other algorithm, presumably one less general than ours, may well solve the extrapolation problem efficiently.

Concluding Observations

The main result of this paper is a family of streams and types for which there is a unified algorithm for extrapolating sequences. We have shown how sequence extrapolation can be treated generally as a problem of learning a stream description.

Our algorithm is not efficient for types that are neither freely generated nor bounded in the number of factorizations of any element. Note that it does not take into account any of the *theory* of the type, i.e., the characterization of its congruence classes—to do so would be type-specific and hence beyond the scope our present objectives. Observing, however, that polynomials of order k are learnable efficiently with only $k+1$ values, we expect that there are interesting cases where incorporating the theory of a type into our algorithm will lead to an efficient extrapolation algorithm for that type. Relating extrapolation to the problem of solving equations over an algebraic data type may be an approach to deriving true lower-bound complexity results on the extrapolation problem for specific types.

Constructive types are a reasonably generous family, but streams over non-constructive types ought to be learnable as well. For example, the family of streams that are polynomial functions with integer coefficients of the stream $\mathbf{N} = \langle 1, 2, 3, \ldots \rangle$ is learnable, but such streams cannot be directly represented as elementary streams over a constructive type because subtraction renders the data type non-constructive. As currently written, the extrapolation algorithm can get into infinite loops for non-constructive types.

Sample-size analysis offers a number of challenging problems. Without the powerful statistical convergence theorems that have been so effective in statistical learning models, we resort to arguing by cases; the resulting proofs quickly become complex and unappetizing. In essence this means we lack any deep understanding about why so many hypotheses are eliminated by so few examples.

The elementary description language itself, while expressive enough for many purposes, is still rather weak. We have limited initial values to atoms in this paper only in order to simplify the reasoning about the algorithm; our implementation does not impose this requirement. By contrast, the block-structure requirement that equality forms may refer back only to their parent streams is necessary for the efficiency of the extrapolation algorithm and of the algorithm for selecting meaningful descriptions from the final induction space. In the course of our research we explored a number of other stream representations that turn out to be equivalent in expressive power to ES, and settled on the one defined in this paper because it yields an efficient algorithm. Still, we see considerable interest in discovering other stream representations with efficient learning algorithms.

Everyone has doubtless experienced the sudden increase in confidence that accompanies discovering a simple rule to account for a lengthy series of observations. Confidence continues to increase rapidly with the number of successful predictions, and at some point confidence in the hypothesis may even exceed one's confidence in the data. As part of this work we developed a Bayesian model of confidence [7] that applies, not just to sequence extrapolation, but to a variety of predictive learning problems. This model leads to an algorithm for sequence extrapolation in the presence of noisy examples. An interesting property of extrapolation from unreliable data is that, unlike statistical learning from independent random events, errors early in the sequence are potentially more significant than errors later on.

Theoretical problems are entertaining, but our motivation in conducting this work is primarily a practical one, and we have implemented the extrapolation algorithm as a first step in exploring several applications. One of these applications is inductive programming, the task of formulating a general concept (function or relation) from examples and counterexamples. An area of active research, inductive programming has been applied to problems in biology, structural engineering, and qualitative modeling, with modest success. Current algorithms for inductive programming are based on finding minimal generalizations of the examples, but these methods require an enormous number of examples and must search through a potentially huge space of possible generalizations. At the same time, they disregard much of the structure available in the examples, structure that is evident if the examples are arranged in order of complexity or size. In many cases that we have studied the number of examples required decreases by two or three orders of magnitude when sequence extrapolation is combined with generalization. Sequence extrapolation also suggests some new approaches to fold-unfold program transformations and to task scheduling in parallel programming on scalable architectures.

Acknowledgments

The authors thank Peter Friedland, Mike Lowry, Steve Minton, David Thompson, and Richard Waters.

References

[1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proc. 18th Symposium on Theory of Computing*, pages 273–282. ACM, 1986.

[2] A. K. Dewdney. Computer recreations. *Scientific American*, pages 14–21, 1986.

[3] T. Dietterich and R. Michalski. Learning to predict sequences. In R. S. Michalski et al., editor, *Machine Learning: An AI Approach, Vol. II.* Morgan Kaufmann, 1986.

[4] D. Haussler. Generalizing the pac model for neural net and other learning applications. Technical Report UCSC-CRL-89-30, University of California, Santa Cruz, 1990.

[5] C. Hedrick. Learning production systems from examples. *Artificial Intelligence*, 7:21–49, 1976.

[6] K. Kotovsky and H. Simon. Empirical tests of a theory of human acquision of concepts for sequential patterns. *Cognitive Psychology*, 4:399–424, 1973.

[7] P. Laird. Weighing hypotheses: Incremental learning from noisy data. Technical report, NASA Ames Research Center, AI Research Branch, 1993. (to be available online from AAAI).

[8] R. Michalski, H. Ko, and K. Chen. Qualitative prediction: the SPARC/G methodology for inductively describing and predicting discrete processes. In *Current Issues in Expert Systems*. Academic Press, 1987.

[9] S. Persson. *Some Sequence Extrapolation Programs: A study of representation and modeling in inquiry system.* PhD thesis, University of California, Berkeley, 1966. Also printed as Stanford University Computer Science Department Technical Report # CS50, 1966.

[10] M. Pivar and M. Finkelstein. Automation, using LISP, of induction on sequences. In E. Berkeley and D. Bobrow, editors, *The Programming Language LISP.* Information International, Inc., 1964.

Appendix: Outline of the Proof of Theorem 8

Let $S = \langle s_1, s_2, \ldots \rangle$ be an $ES(2)$ stream over pairs. We argue first that the description of S is determined up to equivalence by its first $m = h(s_1) + 3$ values. Then we exhibit streams for which this number of values is necessary.

Following is a useful lemma whose proof is not difficult: Let S be a stream of delay at most k; if the first k values of S are all equal, then S is a constant stream.

The proof of the theorem is by induction on the height $h(s_1)$ of the first example. For $h = 0$, assume that the first element s_1 is an atom a. We claim that 3 examples suffice to determine the definition of S up to equivalence—i.e., if there is a stream T of delay at most 2 whose first 3 values are the same as those of S, then any definition for S is equally a definition for T.

Suppose $s_1 = s_2$. Then by the above lemma S is a constant stream and its definition is clearly determined up to equivalence. So assume $s_1 \neq s_2$. The form of the definition is

$$\begin{aligned} S &= \langle a \mid A \rangle \\ A &= \ldots \end{aligned}$$

where the definition of A depends on s_2.

Suppose s_2 is an atom b (possibly equal to a). Then the description of A is $A = b$. (If $b = a$, an equivalent description is $S = a$.)

Suppose s_2 is the pair (a, b), where $a \neq b$. Then the description of A must be a "cons" functional form $A = (B, C)$ where the description of C is necessarily $C = b$. The description of B is either $B = S$ or $B = a$—both descriptions are consistent with the first two examples. The third example will determine which description of B is correct: if s_3 is $((a, b), b)$, then $B = S$, but if s_3 is (a, b), then $B = a$. Hence three examples of the sequence S suffice when s_2 is the pair (a, b). More generally, we can extend this argument by induction on the structure of s_2 to show that the description of A is determined by s_2 and s_3.

Before continuing, let us introduce some notation and terminology. If a stream A is described by the functional form $A = (U, V)$, the *components* of A are A, U, V, and the components of U and V. Such a form can be visualized as a *description tree* whose root is labeled A and whose left and right children are labeled U and V, respectively. If U is in turn a functional form, it, too, has children; but if U is an atom, an equality form, or an initial-value form, then it is a leaf of the tree. In Figure 3, part of a description of a stream S is shown schematically in tree form. The leaf Z is an initial-value form $\langle a \mid A' \rangle$, where the stream A' in turn yields a description tree. Since the delay of S is at most two, the leaves of A' must be either atoms or backreferences to parents of Z in the description tree for S. We refer to A' and its components as *delayed components* of Z (and the parents of Z). We will adopt the convention of priming the names of delayed components. Also, we continue the convention of denoting the examples of a stream X by x_1, x_2, etc. Note that if A' is a delayed component of S, a'_1 (the first instance of A') occurs as part of s_2, and if we have m examples of S, we have only $m - 1$ examples of A'.

Inductive Step. We assume that $h(s_1) + 3$ examples suffice to identify up to equivalence all delay 2 streams such that $h(s_1) < h$.

Let S be a stream of delay 2 such that $h(s_1) = h$. Let $m = h + 3$. Write $s_1 = (\alpha_1, \beta_1)$, where either $h(\alpha_1) = h - 1$ and/or $h(\beta_1) = h - 1$. The description of S is thus of the form $S = (U, V)$, where U and V may likewise be functional forms.

Let us consider the description of U. The description is a functional form $U = (W, X)$ iff α_1 is a pair. In fact, the description of S can be represented as a component tree terminating in a non-functional "leaf" stream Z for each atom a in s_1. These non-functional forms are either initial-value forms $Z = \langle a \mid A' \rangle$ or constant forms, $Z = a$.

Consider any leaf Z corresponding to an atom a in s_1. Let z_2 be the pair (or atom) in s_2 in the position corresponding to this a. If the expression s_1 does not occur as part of z_2, then the description of the stream Z (including its tailstream) cannot contain any backreferences to S. Moreover, if this is true for *all* leaf streams in the description of U—i.e., that they do not contain any backreferences to S because the corresponding value in s_2 does not contain any occurrences of s_1—then U is a stream of delay 2 whose first value has height less than h and whose description depends only upon U and its substreams. Hence by the inductive hypothesis the description of U is determined by only $m - 1$ examples.

In particular, if there exists a description of U without backreferences to S consistent with all m examples, then that description is determined up to equivalence by the first $m - 1$ examples

Therefore let us assume that there is at least one leaf component Z of U whose first value z_1 is an atom a and whose second value z_2 contains within it s_1. The description of Z must be $Z = \langle a \mid A' \rangle$, followed by the description of A'. In general, the description of A' will

$$\begin{array}{rcll}
s_1 & = & ((a,a),(a,a)) & \\
s_2 & = & ((s_1,s_1),(s_1,s_1)) & \equiv (\alpha_2,\beta_2) \\
s_3 & = & ((s_2,(\alpha_2,(a,a))),(s_2,(\beta_2,(a,a)))) & \equiv (\alpha_3,\beta_3) \equiv ((\alpha_{3,1},\alpha_{3,2}),(\beta_{3,1},\beta_{3,2})) \\
s_4 & = & ((s_3,(\alpha_3,(a,a))),(s_3,((\beta_{3,2},\beta_{3,2}),(a,a)))). &
\end{array}$$

Figure 3: Stream with $h = 2$ requiring 5 examples.

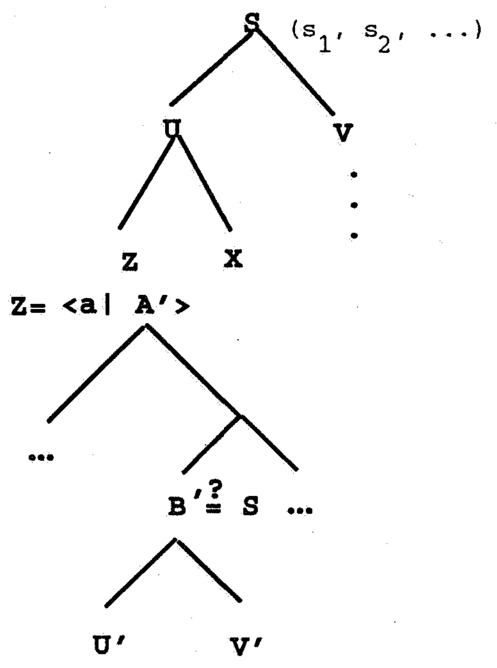

Figure 2: Structure of S.

be a "tree" of functional forms, but we know that at least one component B' of A' (or perhaps A' itself) satisfies $b'_1 = s_1$. Thus $B' = S$ is a hypothetical description of B' consistent with s_1 and s_2; $B' = (U', V')$ may be another hypothesis, where U' and V' are the names for the left and right components of B, respectively. According to our assumptions, $u'_1 = u_1$ and $v'_1 = v_1$.

In m examples of S there are $m - 1$ examples of the delayed component B'. If these $m-1$ examples are sufficient to eliminate either of the two hypotheses for B', then we have no problem. Nor do we have a problem if these hypotheses are equivalent and therefore consistent with $m-1$ examples. Let us consider, however, the possibility that both hypotheses for B' remain consistent with $m - 1$ examples of B', but that later one or the other hypothesis turns out to be inconsistent with an example of B'.

In particular, if, for some i, $1 \leq i \leq m - 1$, either $u'_i \neq u_i$ or $v'_i \neq v_i$, then the hypothesis S for B' will be eliminated because it is inconsistent with the value s_{i+1}. If, moreover, S should be eliminated as a hypothesis for *all* delayed components of U like B', then the only viable hypotheses for U will depend entirely on substreams of U, with no backreferences to S. In this case, by the inductive hypothesis, these $m - 1$ examples suffice to determine U up to equivalence. The only remaining case, therefore, is where $u'_i = u_i$ and $v'_i = v_i$ for all $i \leq m - 1$, for some delayed components of U like B'. Thus we need to show that if, after m examples of S, we still have two consistent hypotheses for B', then they must be equivalent.

The consistent hypotheses for U without backreferences to S are determined uniquely up to equivalence by $m-1$ examples. Part of such a hypothesis is the description $B' = (U', V')$. Since $U' = U$ is consistent with the examples, any hypothesis for U' is equivalent to this one. Note that $V' = V$ is not a possible hypothesis, since V is not in the parent hierarchy of V'. However, it can be argued that V' must be equivalent to V. Thus the two hypotheses $B' = S$ and $B' = (U', V')$ are equivalent. Having shown that the description of U is determined (up to equivalence) by m values, we can apply the same argument to V by symmetry. This completes the proof that streams of delay 2 are $m = h + 3$-distinguishable.

To show that they are exactly $h(s_1)+3$-distinguishable, we construct for arbitrary h a sequence $s_1, \ldots, s_{m-1}$ such that there are at least two inequivalent hypotheses consistent with all $m - 1$ examples.

The construction, while not difficult, is too lengthy to give here. We provide as examples the construction for $h(s_1) = 0$ and $h(s_1) = 2$.

For $h(s_1) = 0$: The stream

$$S = \langle a, (a, a), \ldots \rangle$$

has four distinct consistent descriptions:

$$\begin{array}{rcl}
S & = & \langle a \mid A \rangle \\
A & = & (B, C) \\
B & = & S \text{ or } a \\
C & = & S \text{ or } a
\end{array}$$

Hence two examples do not suffice to determine its description.

For $h(s_1) = 2$: The first 4 examples of S are in Figure 3 above. One can verify that S has four distinct consistent descriptions. Hence four examples do not suffice to determine its description.